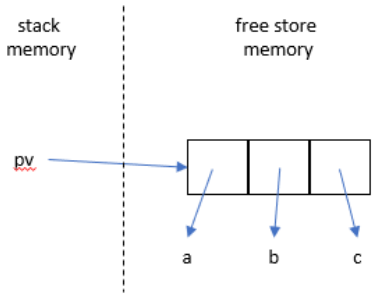
Pointers and Memory Management

In this question, `pv` is a variable of type `vector<string*>*`. The vector was allocated on the free store using `new`, and the pointers in it point to different strings that were allocated on the free store using `new`. There are no null pointers.

a) (2 marks) Draw a diagram that shows what the pointers and vectors looks like in memory. Assume it has 3 strings, "a", "b", and "c". Assume `pv` is on the call stack.

Sample Solution



Marking Scheme

- **+1 mark**: correct pointers
- **+1 mark**: correct memory regions

b) (2 marks) Write a C++ loop that prints to cout each string pointed to by `pv`.

Sample Solution

```
// solution 1: print using for-each loop
for(string* p : *pv) cout << *p << " ";

// solution 2: print using for-loop
for (int i = 0; i < pv->size(); i++) {
    string s = *(*pv)[i];
    cout << s << " ";
}
```

Marking Scheme

- **+1 mark**: correct loop header
- **+1 mark**: correct loop body

Up to -1 mark deducted if the code is very inefficient, or does anything unnecessary.

c) (2 marks) Write a fragment of C++ code that properly de-allocates the vector `p` points to, and also all the strings pointed to by it. There should be no memory leaks or other errors.

Sample Solution

```
for (int i = 0; i < pv->size(); i++) {
    delete (*pv)[i];
}
delete pv;
```

Marking Scheme

- **+1 mark**: deleting strings
- **+1 mark**: deleting vector

Up to -1 mark deducted if the code is very inefficient, or does anything unnecessary.

- **-0.5 marks** for deleting vector before strings
- **-0.5 marks** for wrong delete

Object-oriented Programming and Inheritance

(5 marks) Create a class called `Circle` that stores the center and radius of a circle. Make these private, and call them `x`, `y`, and `radius`. In addition, add the following:

1. A **default constructor** that sets both `x` and `y` to 0, and the `radius` to 100.
2. A **copy constructor** that uses an **initialization list** to make a new `Circle` object that is a copy of a another `Circle` object.
3. A **destructor** that prints "done!".
4. A **setter** that lets the user change the `radius` of the circle. If a user tries to set `radius` to a value that is 0 or less, then the `radius` is *not* changed.

Sample Solution

```
class Circle {
    double x;
    double y;
    double radius;

public:
    Circle() : x(0), y(0), radius(100) {}

    Circle(const Circle& c)
    : x(c.x), y(c.y), radius(c.radius) {}

    ~Circle() { cout << "done!"; }

    void set_radius(double r) {
        if (r > 0) radius = r;
    }
};
```

Marking Scheme

- **+1 mark**: correct private variables
- **+1 mark**: correct default constructor that uses an initialization list
- **+1 mark**: correct copy constructor that uses an initialization list
- **+1 mark:** correct destructor
- **+1 mark:** correct setter

Up to -1 mark deducted if the code is very inefficient, or does anything unnecessary.

Notes

- If an error is very small, or you maybe just a slip of the pen, you might decide to take no marks off, or only a few marks. Forgetting one ";" is probably not worth taking any marks off. But repeatedly forgetting a ";" should have a deduction of at least 0.5.
- Sometimes answers can be different than what the marking scheme expects, and in that case you may need to "wing it". In such cases, first try to decide if it is a failing or passing answer. If it's failing, don't give more than 50%.

Multiple Choice Answers
1. A
2. D
3. A